CS391L: Machine Learning WB

Language Modeling: RNNs, Attention and Transformers

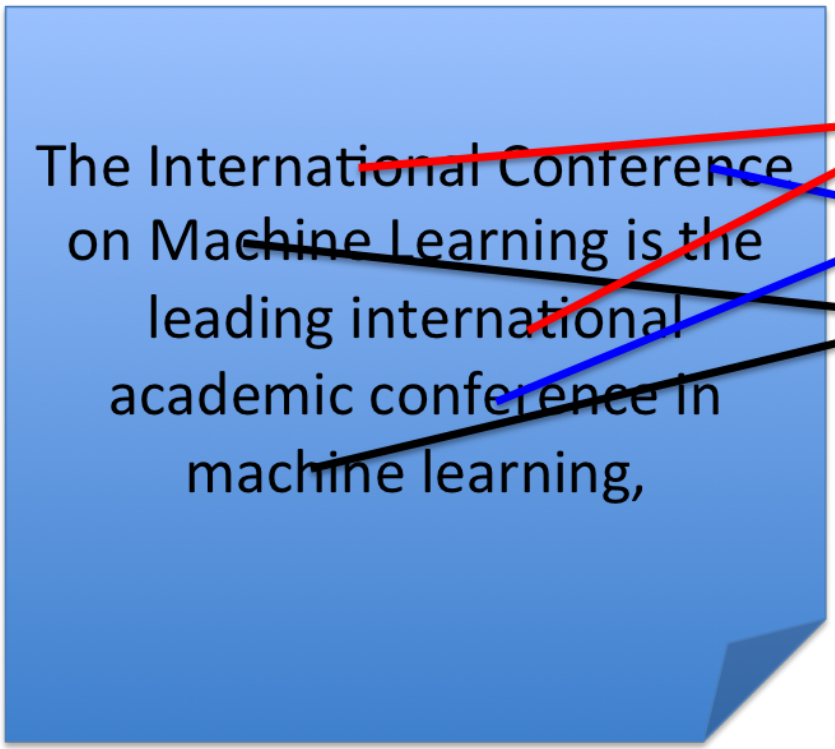
Inderjit Dhillon
UT Austin

April 1, 2025

Representation for sentence/document

Bag of Word

- A classical way to represent NLP data
- Each sentence (or document) is represented by a d-dimensional vector $\boldsymbol{x}$, where x_i is number of occurrences of word i.



(international)	2
(conference)	2
(machine)	2
(train)	0
(learning)	2
(leading)	1
(totoro)	0

number of features = number of potential words (very large)

Feature generation for documents

- Bag of n-gram features ($n = 2$):

The International Conference on Machine Learning is the leading international academic conference in machine learning,

(international)	2
(conference)	2
(machine)	2
(train)	0
(learning)	2
(leading)	1
(totoro)	0

(international conference)	1
(machine learning)	2
(leading international)	1
(totoro tiger)	0
(tiger woods)	0
(international academic)	1
(international academic)	1

10,000 words $\Rightarrow 10,000^2$ potential features

Data Matrix (document)

- Use the bag-of-words matrix or the normalized version (TF-IDF) for a dataset (denoted by D):

$$\text{tfidf}(\text{doc}, \text{word}, \text{D}) = tf(\text{doc}, \text{word}) \cdot idf(\text{word}, \text{D})$$

  - tf (doc, word): term frequency

    (word count in the document)/(total number of terms in the document)

  - idf (word, Dataset): inverse document frequency
    $1 + \log((\text{Number of documents})/(\text{Number of documents with this word}))$

	angeles	los	new	post	times	york
d1	0	0	1	0	1	1
d2	0	0	1	1	0	1
d3	1	1	0	0	1	0

tf-idf

	angeles	los	new	post	times	york
d1	0	0	0.584	0	0.584	0.584
d2	0	0	0.584	1.584	0	0.584
d3	1.584	1.584	0	0	0.584	0

Bag of words + linear model

- Example: text classification (e.g., sentiment prediction, review score prediction)
- Linear model: $y \approx \text{sign}(\boldsymbol{w}^T \boldsymbol{x})$

  (e.g., by linear SVM/logistic regression)
- w_i: the "contribution" of each word

Bag of words + Fully connected network

- $f(\boldsymbol{x}) = W_L \sigma(W_{L-1} \cdots \sigma(W_0 \boldsymbol{x}))$
- The first layer W_0 is a $d_1 \times d$ matrix:
  - Each column $\boldsymbol{w}_i$ is a d_1-dimensional representation of i-th word (word embedding)
  - $W_0 \boldsymbol{x} = x_1 \boldsymbol{w}_1 + x_2 \boldsymbol{w}_2 + \cdots + x_d \boldsymbol{w}_d$ is a linear combination of these vectors
  - W_0 is also called the word embedding matrix
  - Final prediction can be viewed as an $L-1$ layer network on $W_0 \boldsymbol{x}$ (average of word embeddings)
- Not capturing the sequence information in language

Recurrent Neural Network

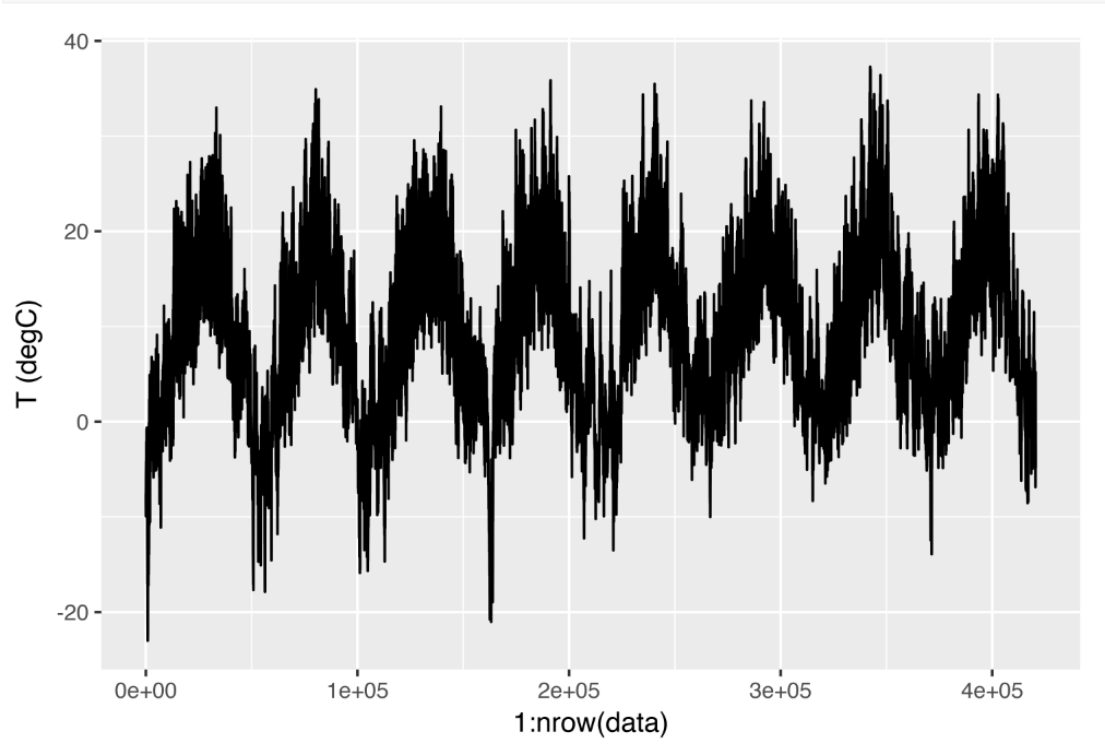
Time Series/Sequence Data

- Input: $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_T\}$
  - Each $\boldsymbol{x}_t$ is the feature at time step t
  - Each $\boldsymbol{x}_t$ can be an d-dimensional vector
- Output: $\{y_1, y_2, \cdots, y_T\}$
  - Each y_t is the output at step t
  - Multi-class output or Regression output:

$$y_t \in \{1, 2, \cdots, L\} \quad \text{or} \quad y_t \in \mathbb{R}$$

Example: Time Series Prediction

- Climate Data:
  - $\boldsymbol{x}_t$: temperature at time t
  - y_t: temperature (or temperature change) at time $t+1$
- Stock Price: Predicting stock price



Example: Language Modeling

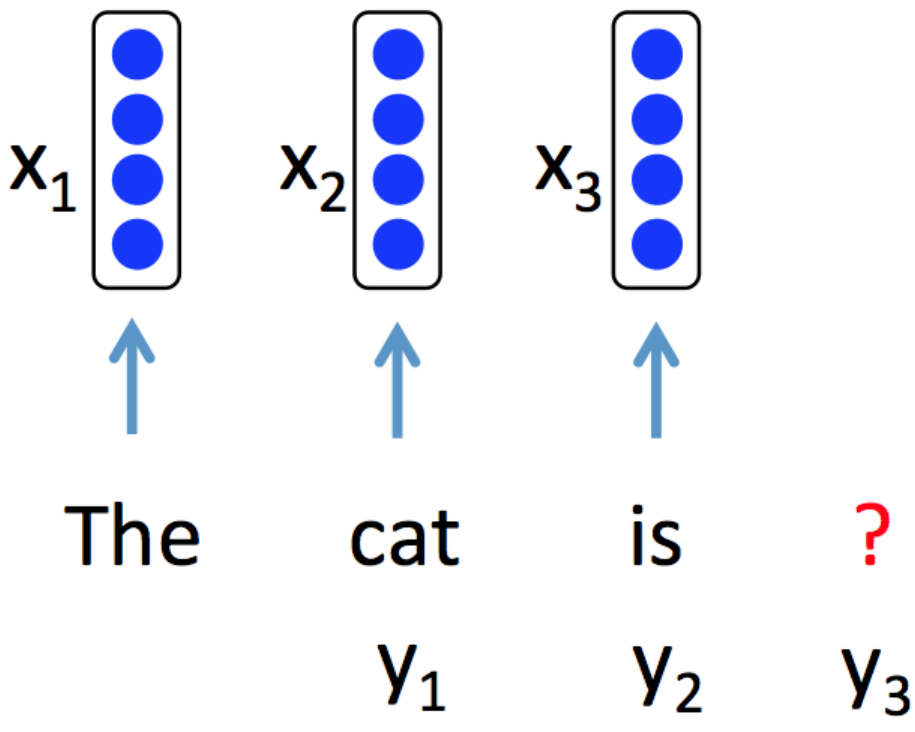
The cat is ?

Example: Language Modeling

The cat is ?

- $\boldsymbol{x}_t$: one-hot encoding to represent the word at step t ($[0, \ldots, 0, 1, 0, \ldots, 0]$)
- y_t: the next word

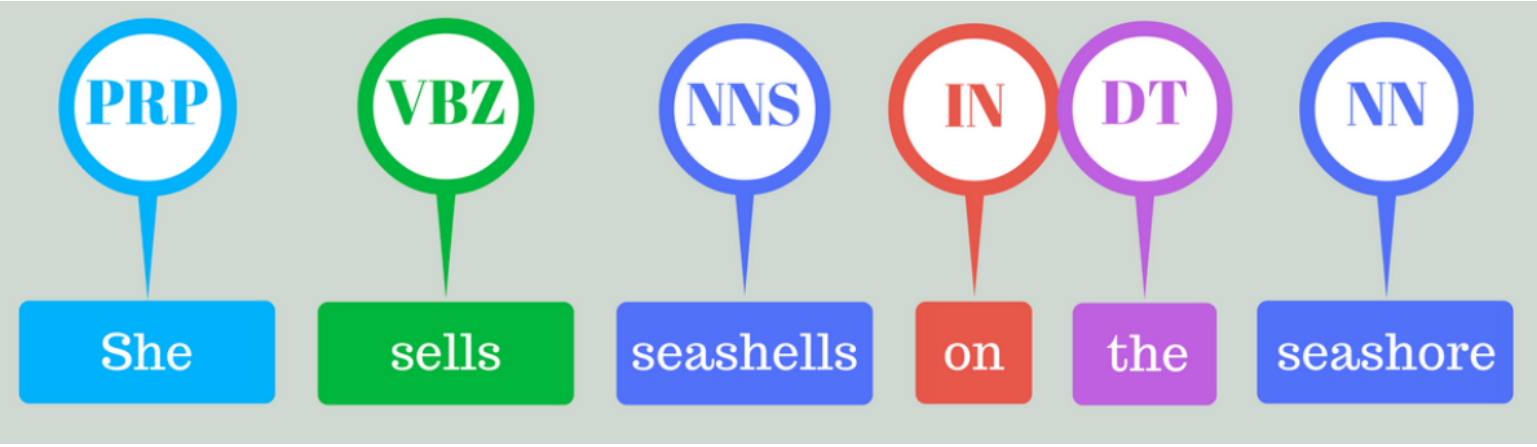
$y_t \in \{1, \cdots, V\}$ V: Vocabulary size



Example: POS Tagging

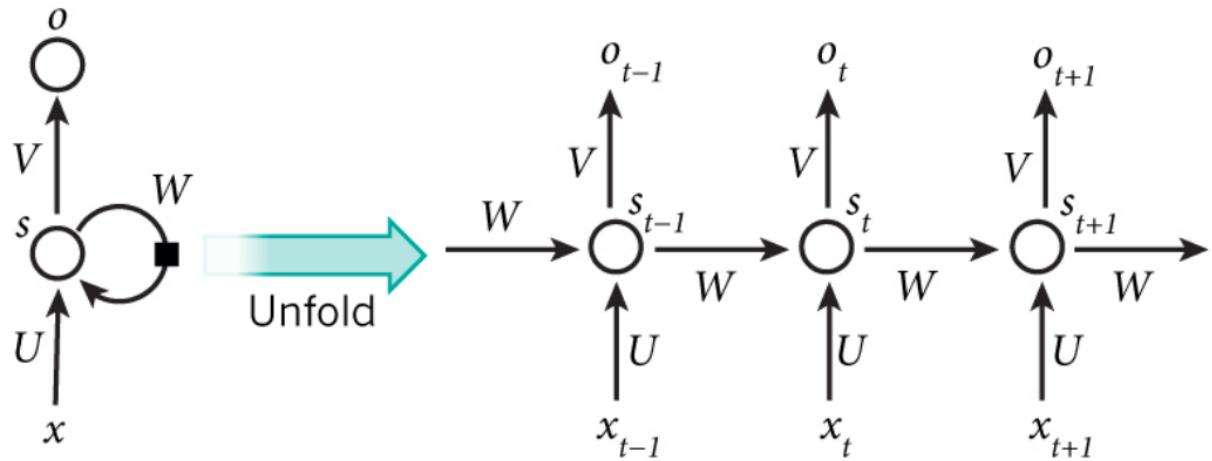
- Part of Speech Tagging:

  Labeling words with their Part-Of-Speech (Noun, Verb, Adjective, $\cdots$)
- $\boldsymbol{x}_t$: a vector to represent the word at step t
- y_t: label of word t



picture from `https://medium.com/analytics-vidhya/pos-tagging-using-conditional-random-fields-92077e5eaa31`

Recurrent Neural Network (RNN)



- $\boldsymbol{x}_t$: t-th input
- $\boldsymbol{s}_t$: hidden state at time t ("memory" of the network)

$$\boldsymbol{s}_t = f(U\boldsymbol{x}_t + W\boldsymbol{s}_{t-1})$$

W: transition matrix, U : word embedding matrix
$\boldsymbol{s}_0$ usually set to be 0

- Predicted output at time t:

$$o_t = \arg\max_i (V\boldsymbol{s}_t)_i$$

Recurrent Neural Network (RNN)

- Training: Find U, W, V to minimize empirical loss:
- Loss of a sequence:

$$\sum_{t=1}^{T} \text{loss}(V\boldsymbol{s}_t, y_t)$$

  ($\boldsymbol{s}_t$ is a function of U, W, V)
- Loss on the whole dataset:

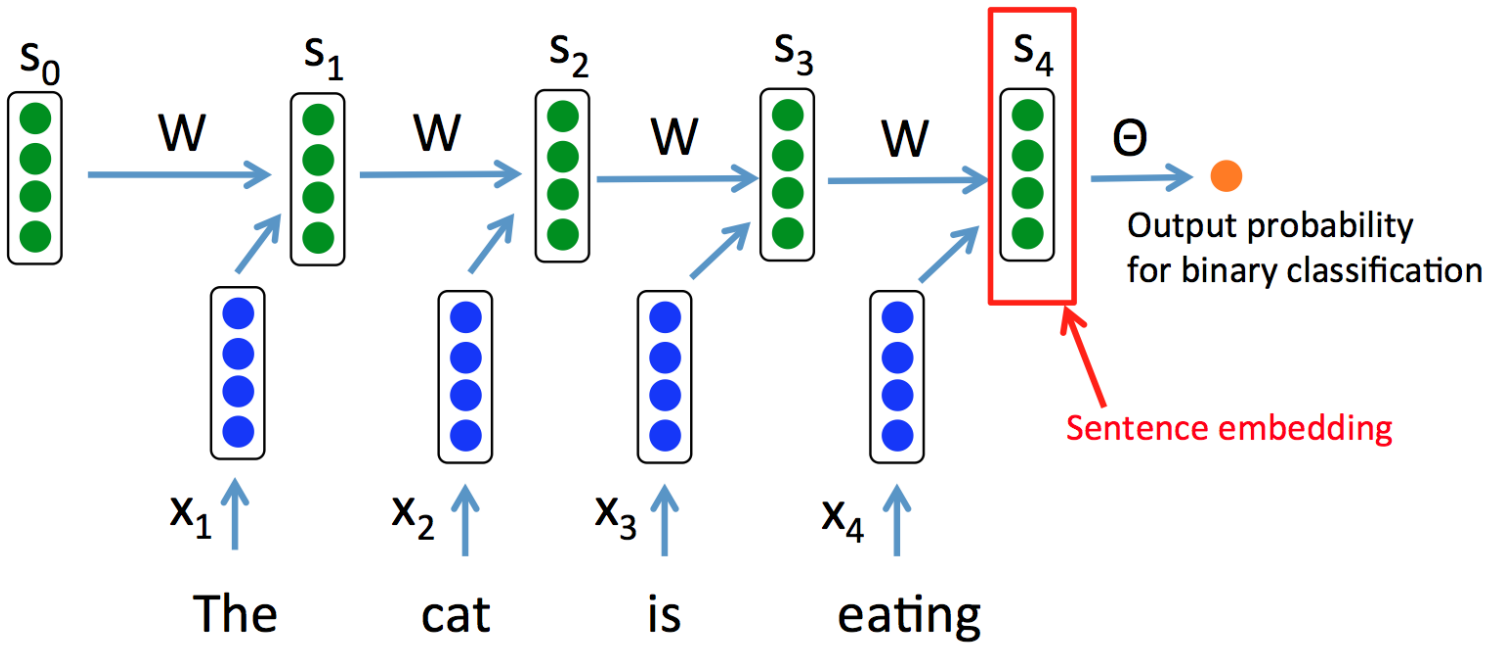
  Average loss over all sequences
- Solved by SGD/Adam

RNN: Text Classification

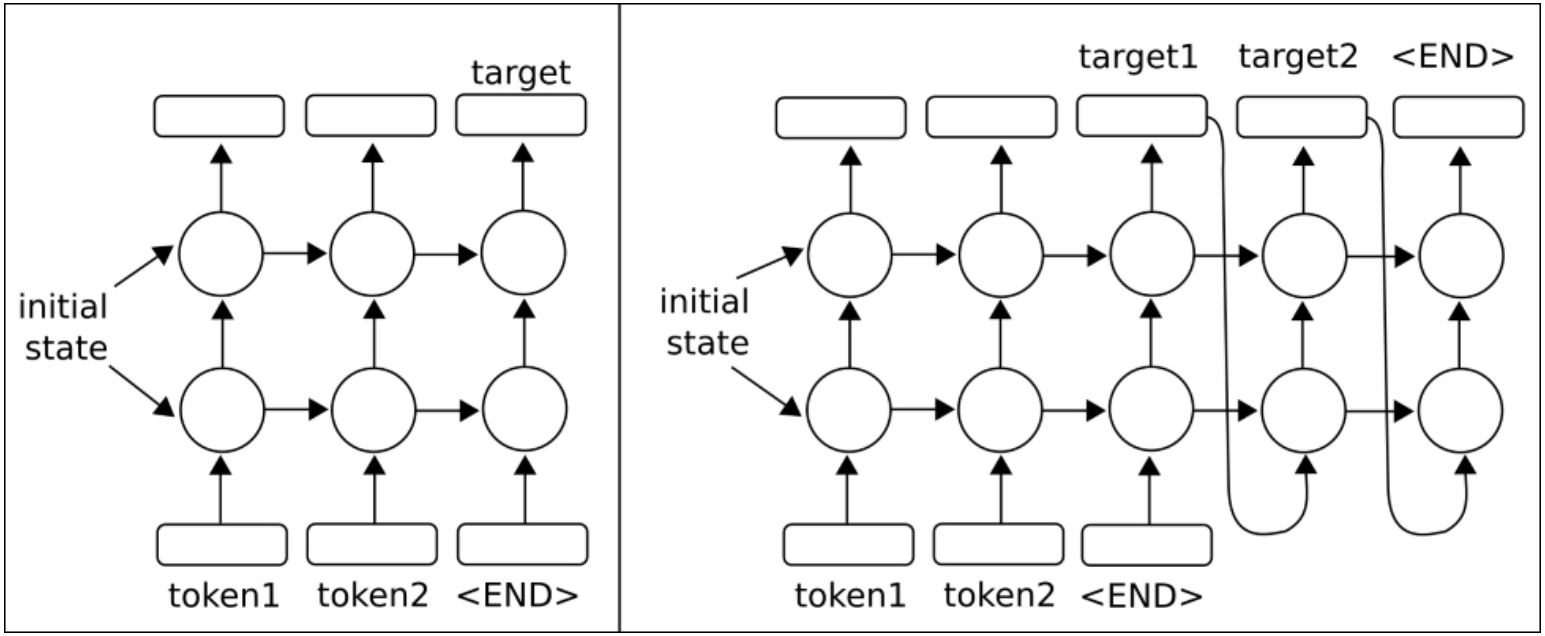
- Not necessary to output at each step
- Text Classification:

$$\text{sentence} \rightarrow \text{category}$$

  Output only at the final step
- Model: add a fully connected network to the final embedding



Multi-layer RNN



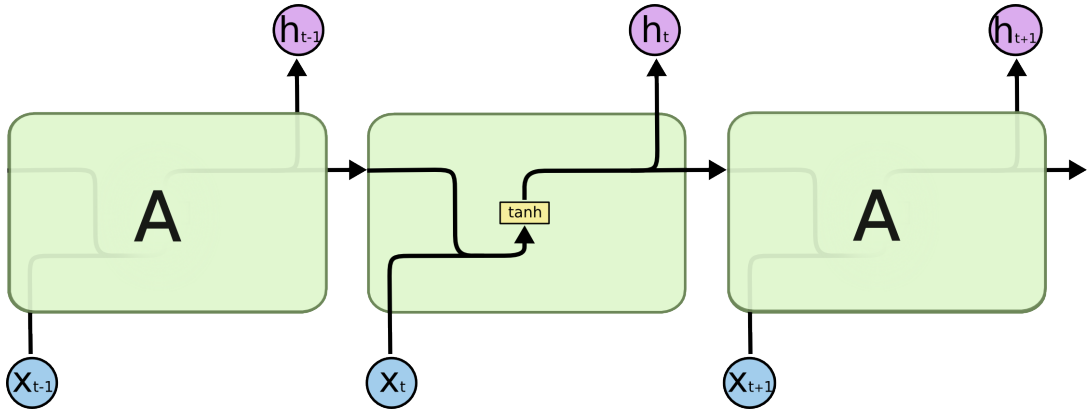
(Figure from
`https://subscription.packtpub.com/book/big_data_and_business_intelligence`)

Problems with Classical RNN

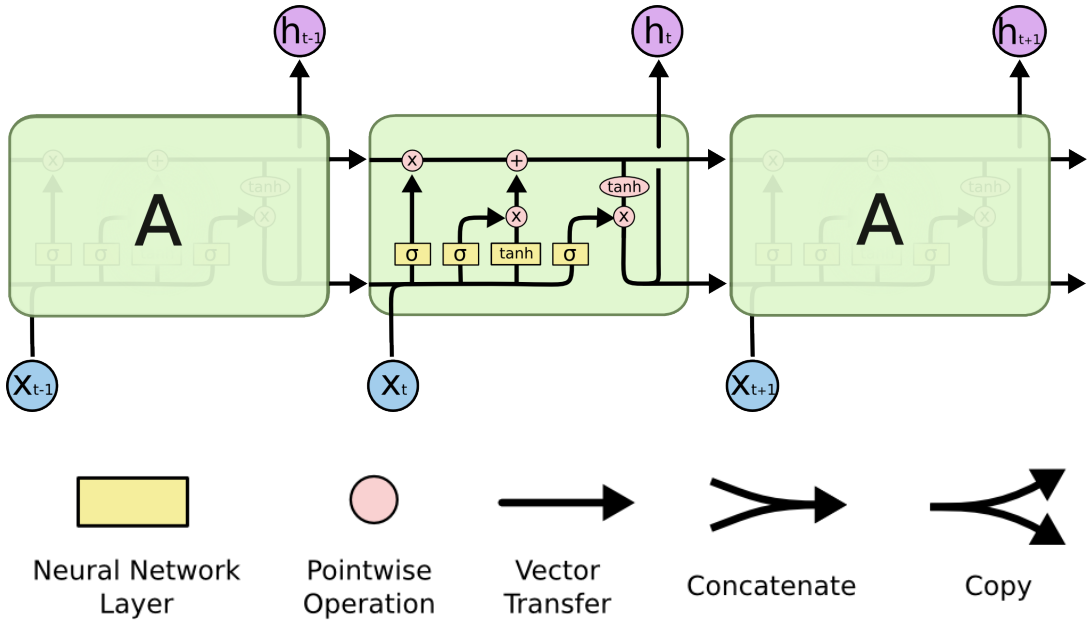
- Hard to capture long-term dependencies
- Hard to solve (vanishing gradient problem)
- Solution:
  - LSTM (Long Short Term Memory networks)
  - GRU (Gated Recurrent Unit)
  - ...

LSTM

- RNN:

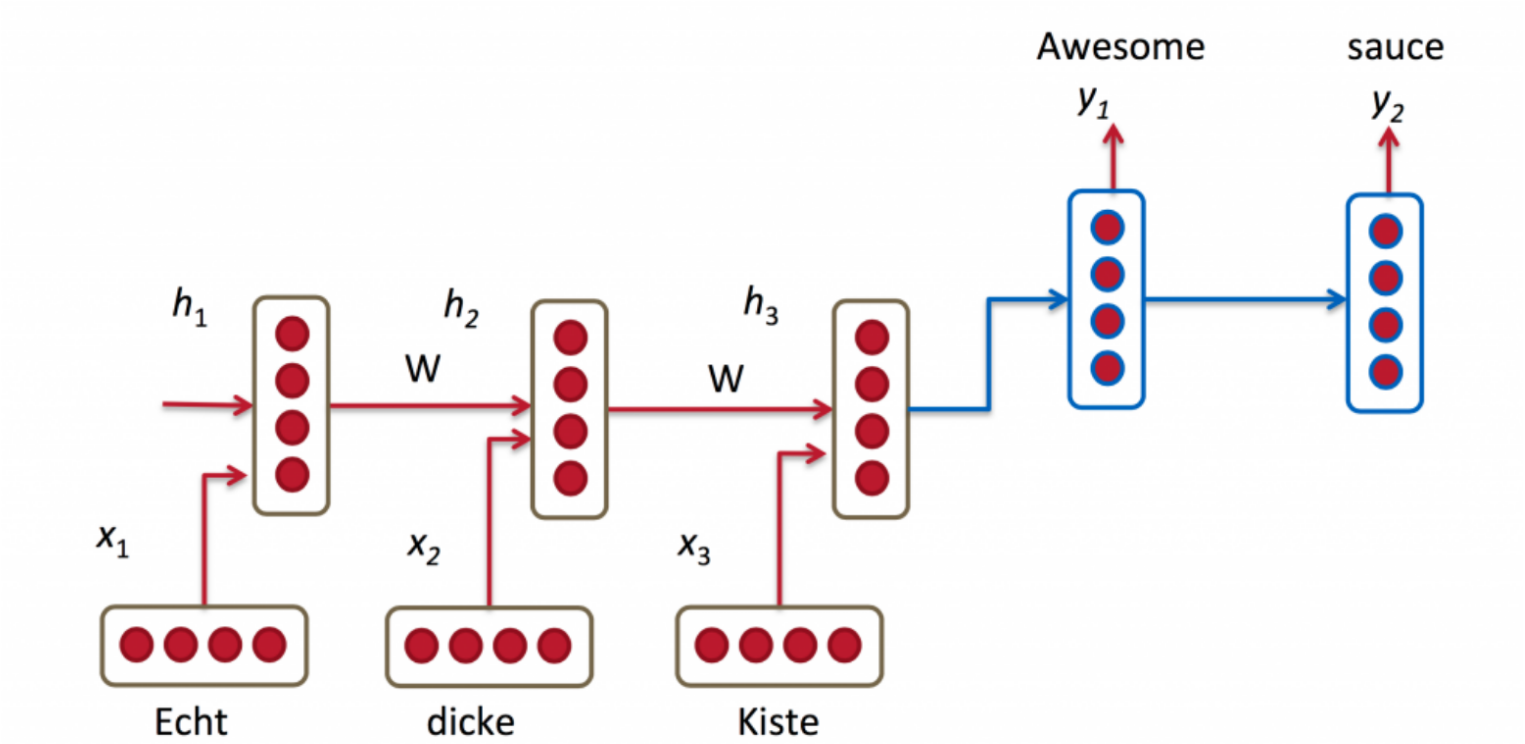


- LSTM:

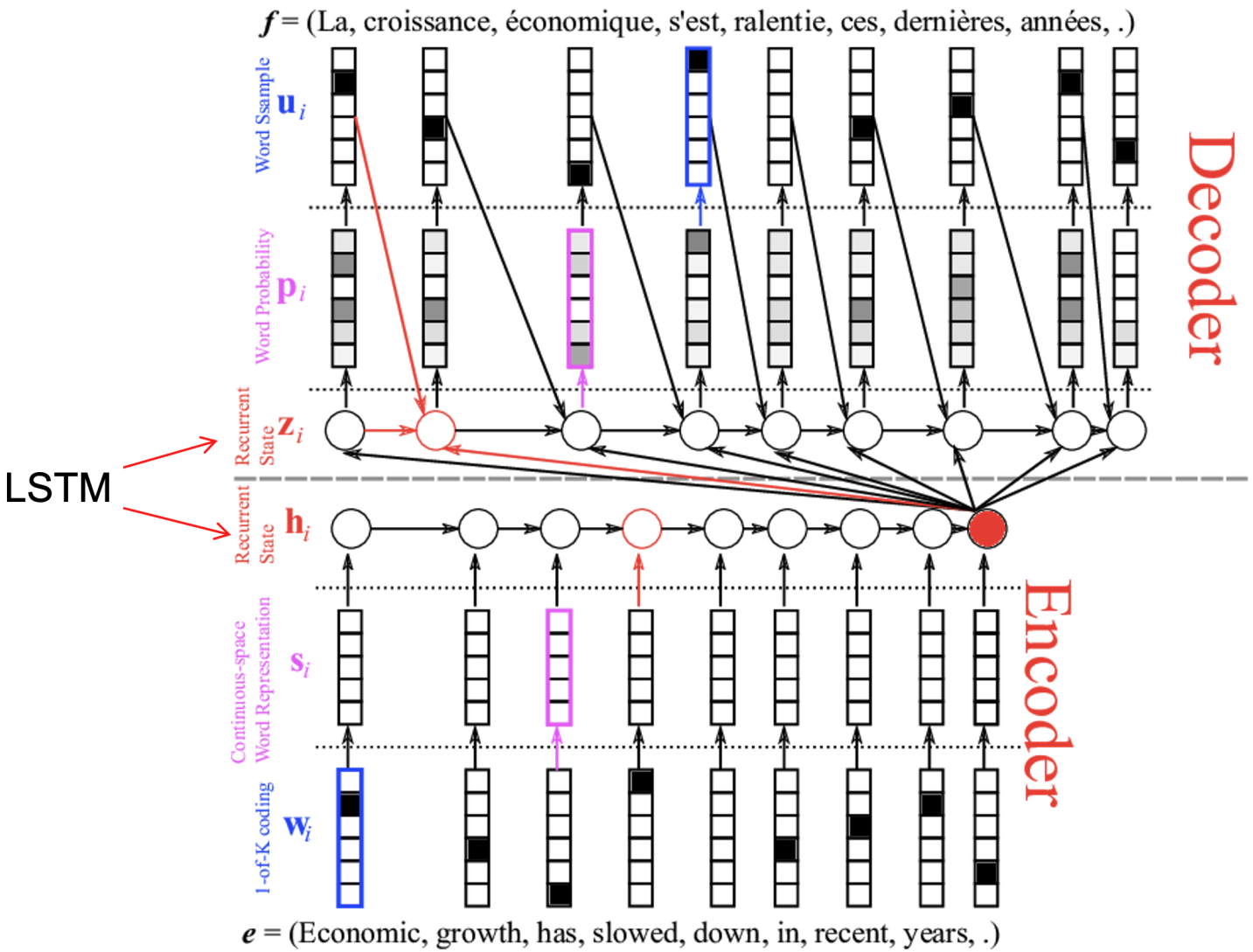


Neural Machine Translation (NMT)

- Output the translated sentence given an input sentence
- Training data: a set of input-output pairs (supervised setting)
- Encoder-decoder approach:
  - Encoder: Use (RNN/LSTM) to encode the input sentence input a latent vector
  - Decoder: Use (RNN/LSTM) to generate a sentence based on the latent vector



Neural Machine Translation



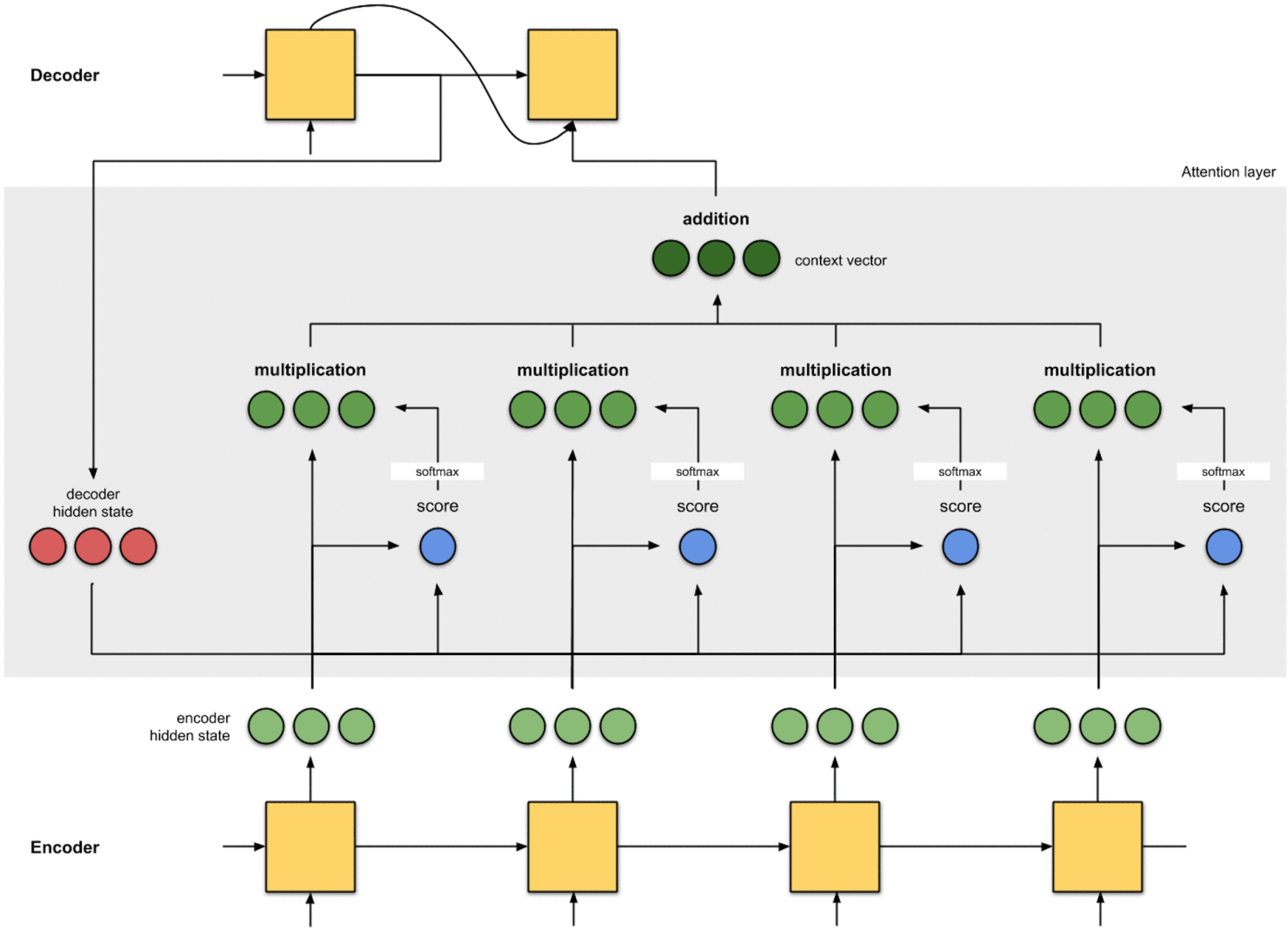
Attention in NMT

- Usually, each output word is only related to a subset of input words (e.g., for machine translation)
- Let $\boldsymbol{u}$ be the current decoder latent state
  $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n$ be the latent sate for each input word
- Compute the weight of each state by

$$\boldsymbol{p} = \text{Softmax}(\boldsymbol{u}^T \boldsymbol{v}_1, \ldots, \boldsymbol{u}^T \boldsymbol{v}_n)$$

- Compute the context vector by $V\boldsymbol{p} = p_1 \boldsymbol{v}_1 + \cdots + p_n \boldsymbol{v}_n$

Attention in NMT

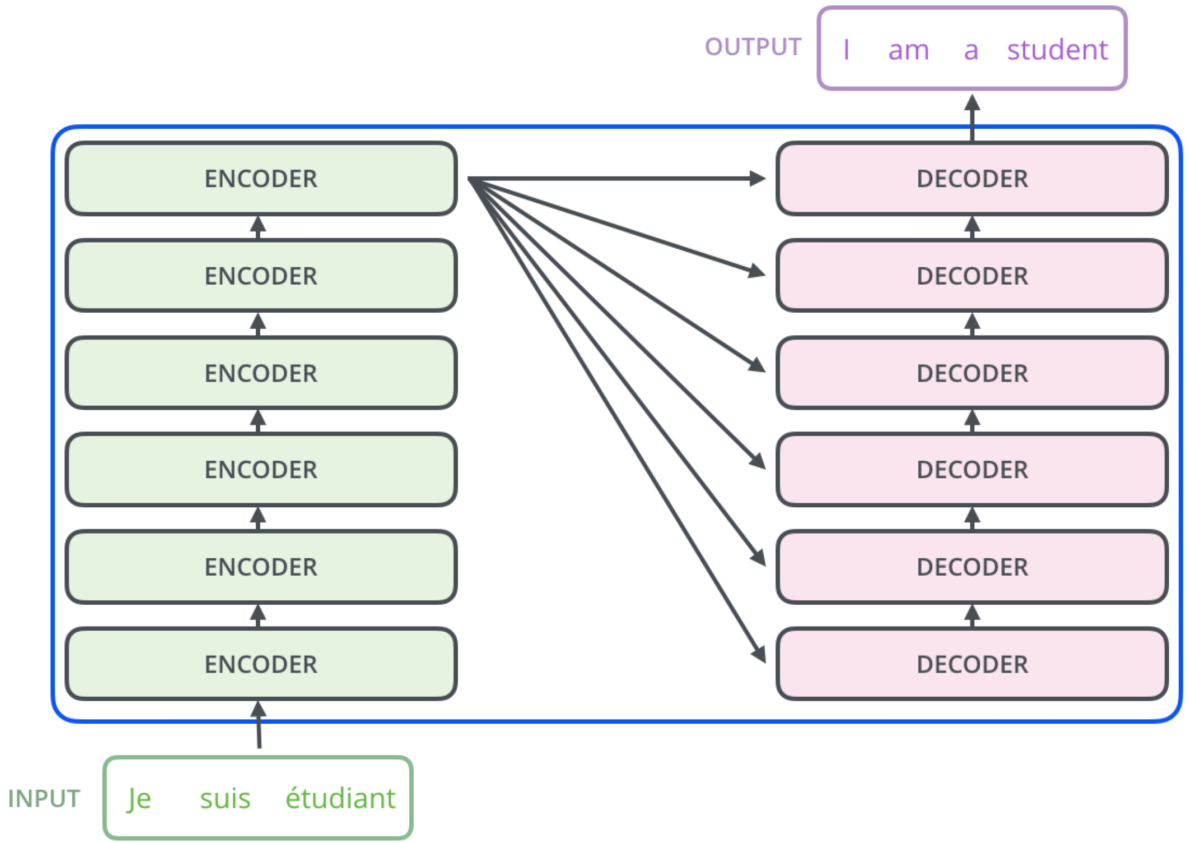


(Figure from `https://towardsdatascience.com/`

Transformer

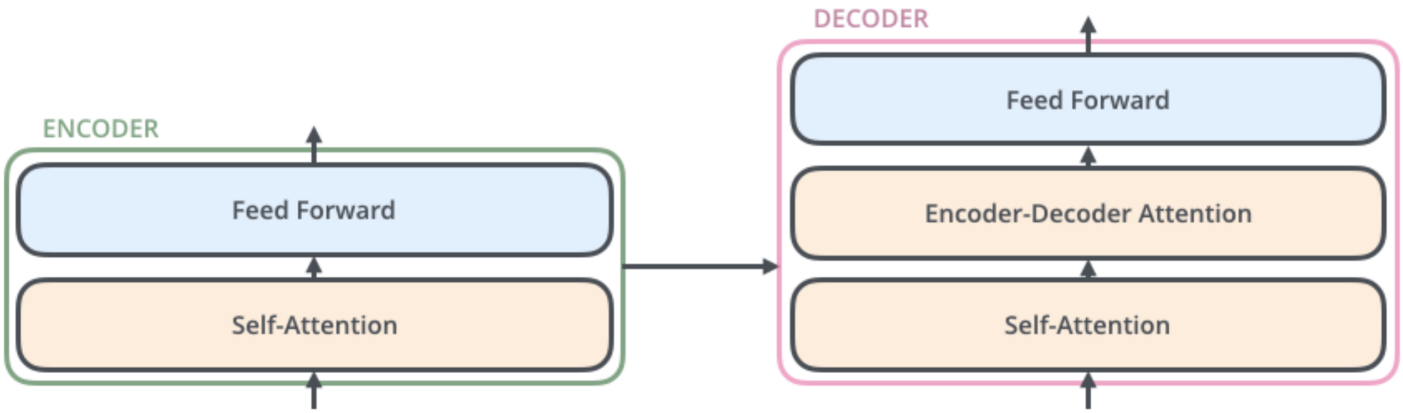
Transformer

- An architecture that replies entirely on attention without using CNN/RNN
- Proposed in "Attention Is All You Need" (Vaswani et al., 2017)
- Initially used for neural machine translation



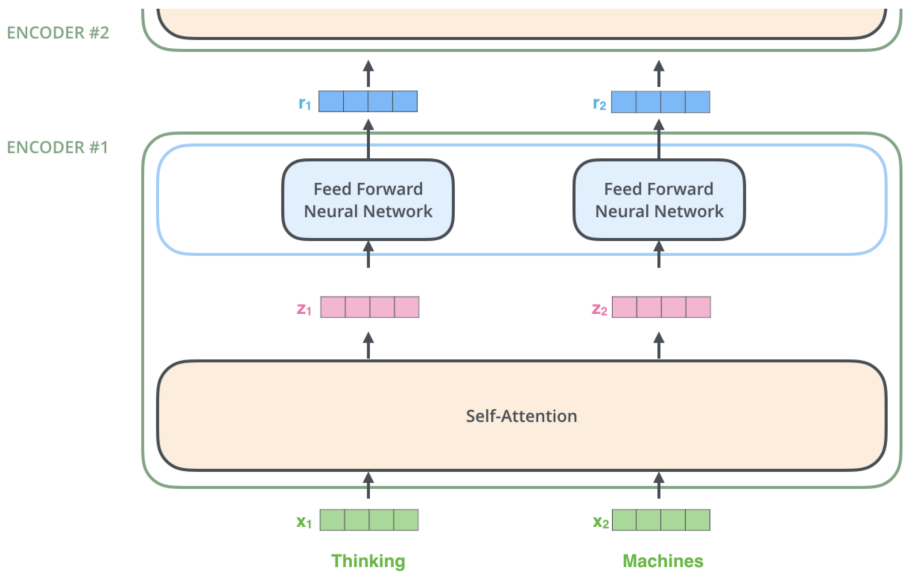
Encoder and Decoder

- Self attention layer: the main architecture used in Transformer
- Decoder: will have another attention layer to help it focus on relevant parts of the input sentence.



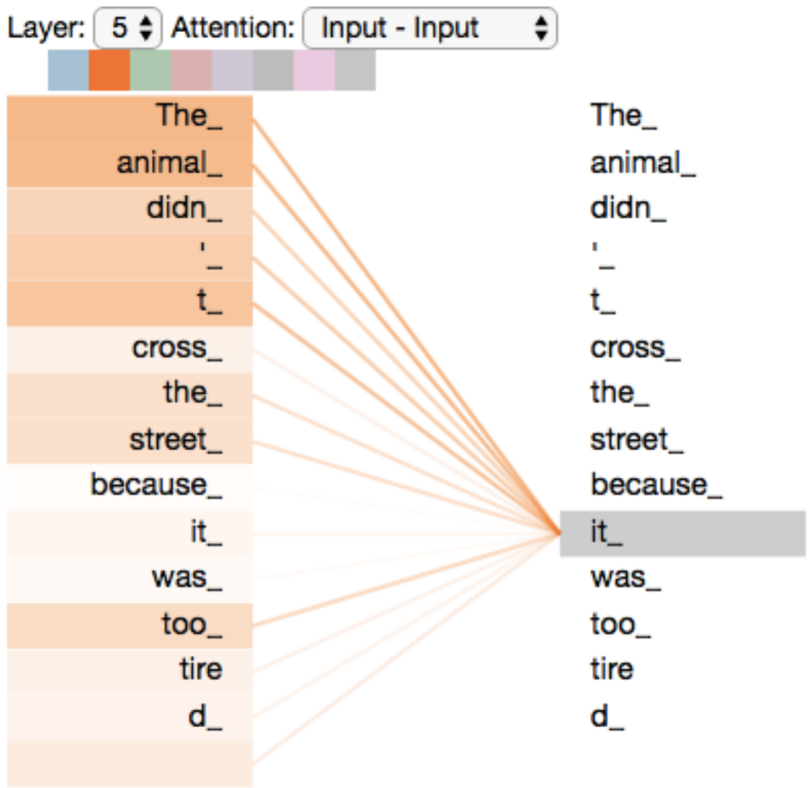
Encoder

- Each word has a corresponding "latent vector" (initially the word embedding for each word)
- Each layer of encoder:
  - Receive a list of vectors as input
  - Passing these vectors to a self-attention layer
  - Then passing them into a feed-foward layer
  - Output a list of vectors



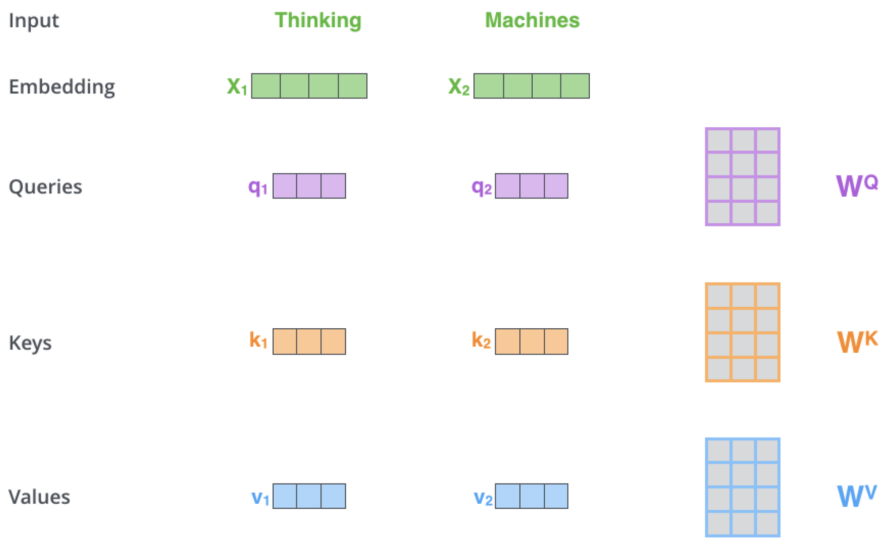
Self-attention layer

- Main idea: The actual meaning of each word may be related to other words in the sentence
- The actual meaning (latent vector) of each word is a weighted (attention) combination of other words (latent vectors) in the sentences



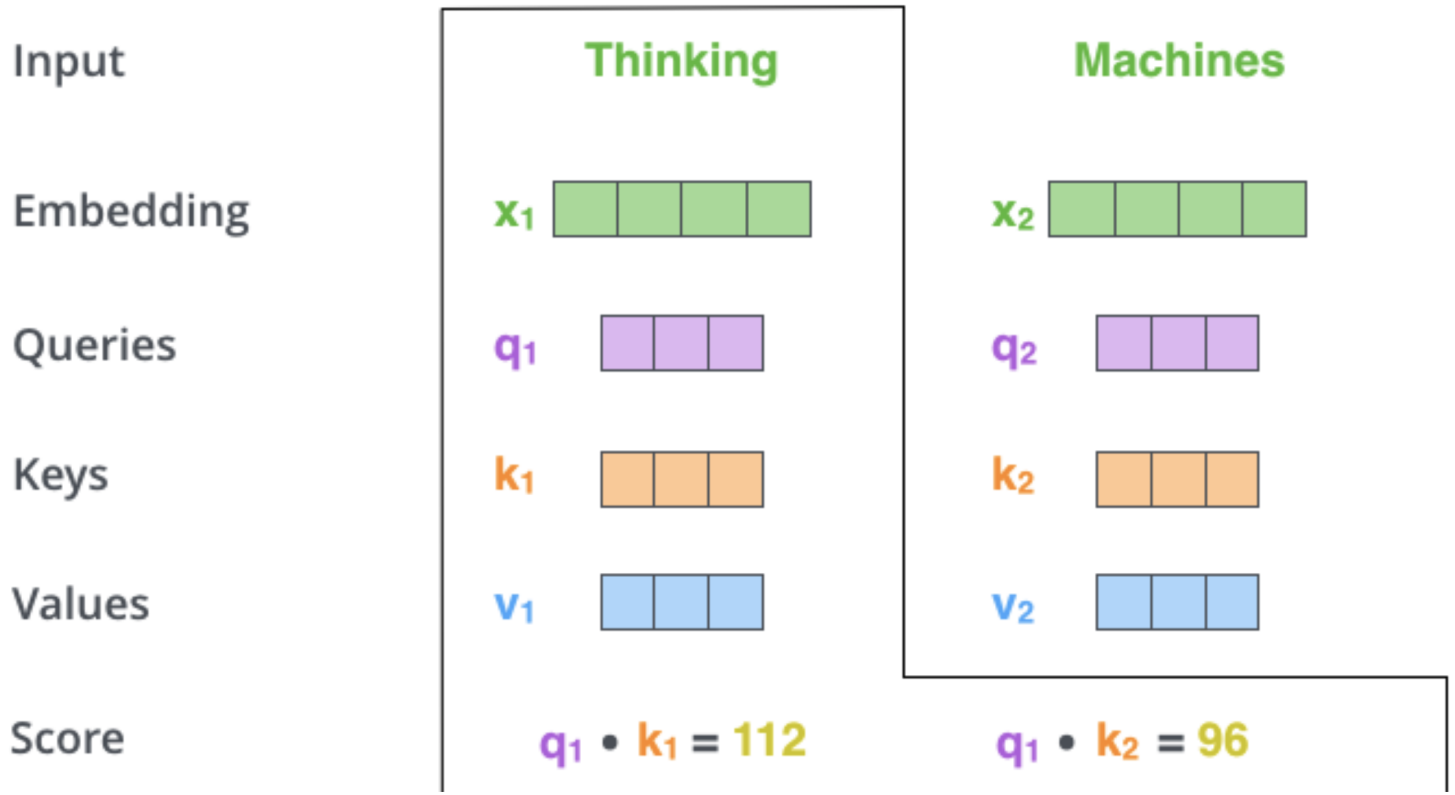
Self-attention layer

- Input latent vectors: $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$
- Self-attention parameters: W^Q, W^K, W^V (weights for query, key, value)
- For each word i, compute
  - Query vector: $\boldsymbol{q}_i = \boldsymbol{x}_i W^Q$
  - Key vector: $\boldsymbol{k}_i = \boldsymbol{x}_i W^K$
  - Value vector: $\boldsymbol{v}_i = \boldsymbol{x}_i W^V$



Self-attention layer

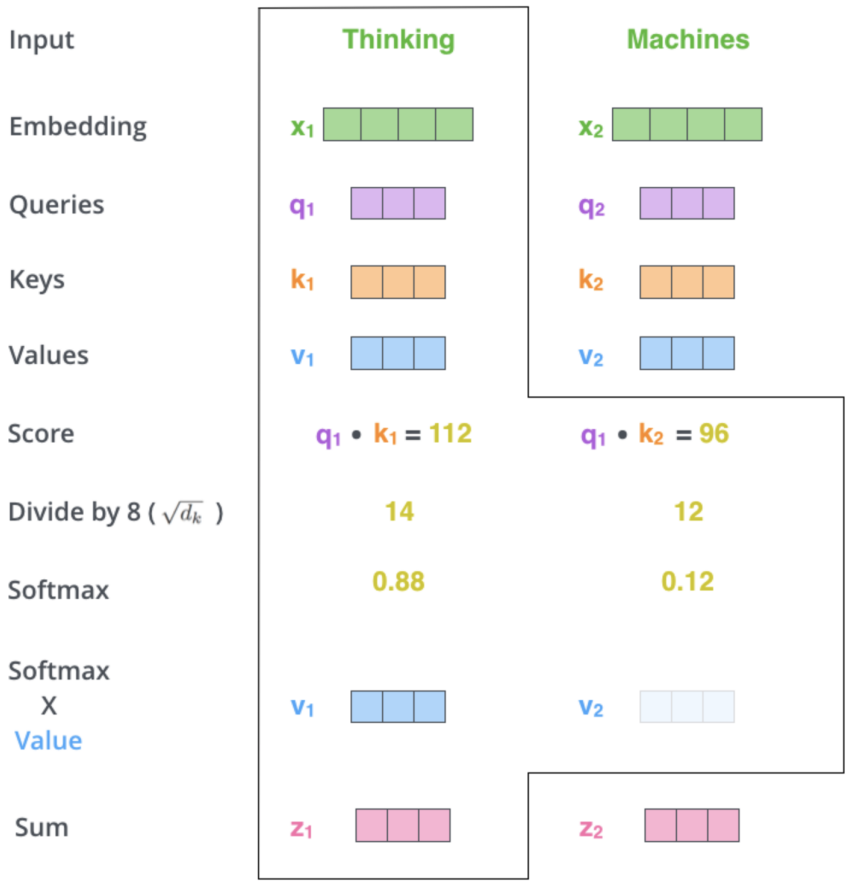
- For each word i, compute the scores to determine how much focus to place on other input words
  - The attention score for word j to word i: $\boldsymbol{q}_i^T \boldsymbol{k}_j$



Self-attention layer

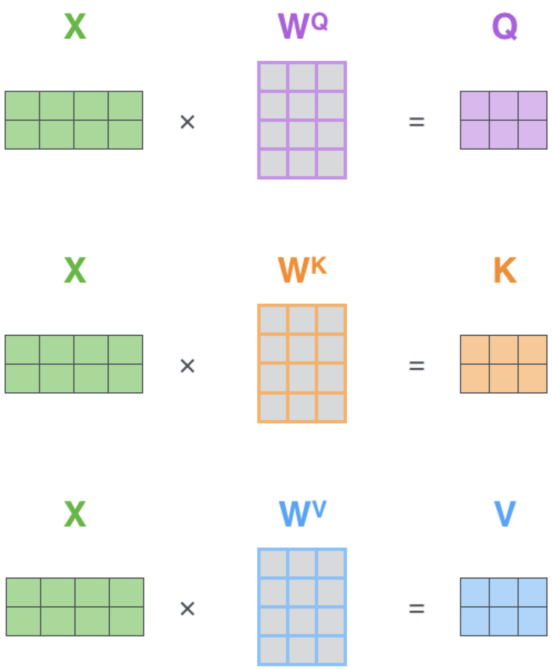
- For each word i, the output vector

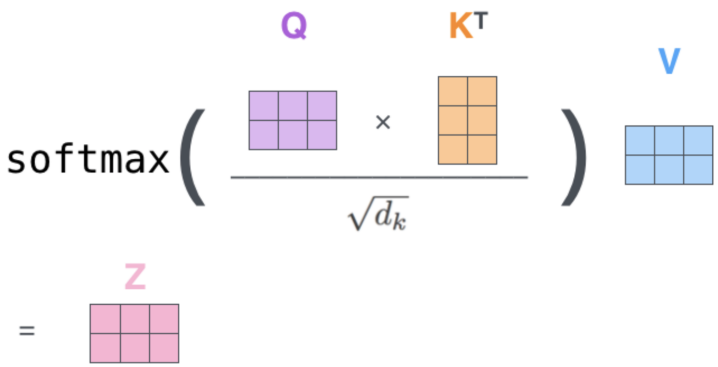
$$\sum_j s_{ij} \boldsymbol{v}_j, \quad \boldsymbol{s}_i = \text{softmax}(\boldsymbol{q}_i^T \boldsymbol{k}_1, \ldots, \boldsymbol{q}_i^T \boldsymbol{k}_n)$$



Matrix form

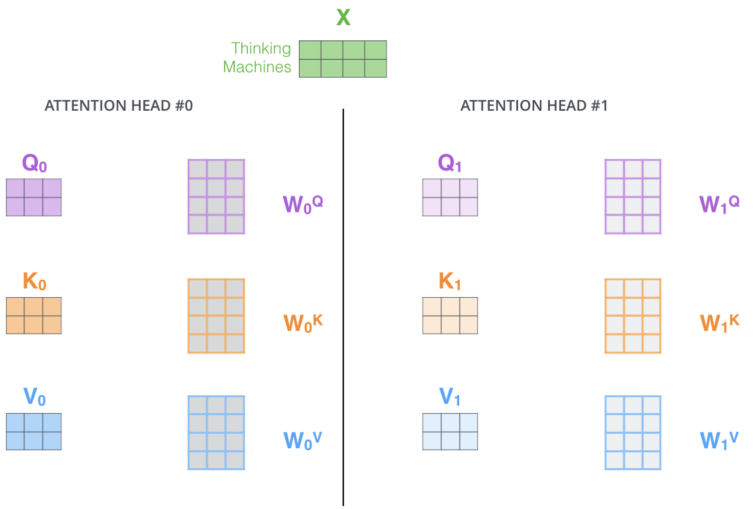
$$Q = XW^Q, \; K = XW^K, \; V = XW^V, \quad Z = \text{softmax}(QK^T)V$$

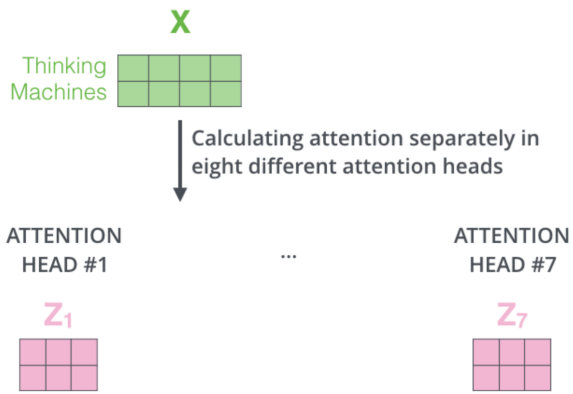




Multiple heads

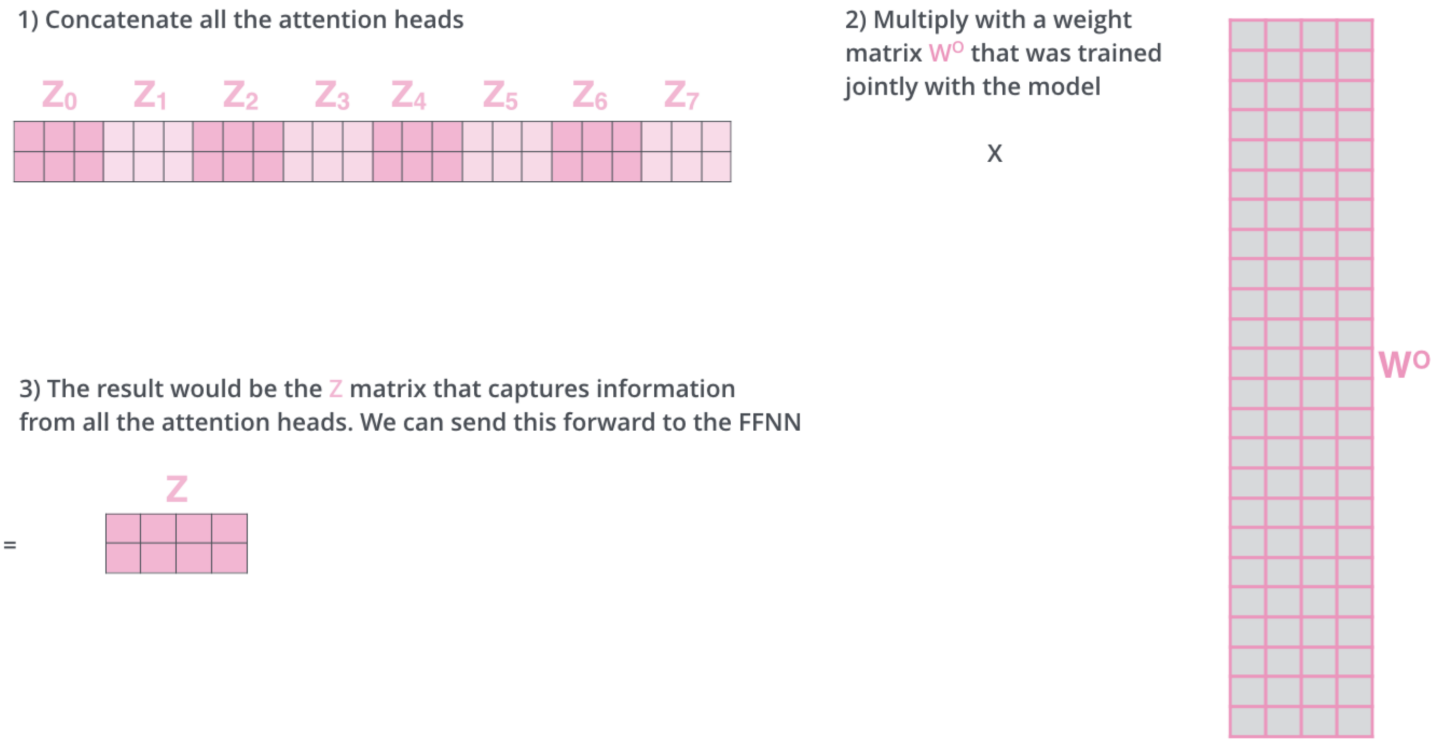
- Multi-headed attention: use multiple set of (*key*, *value*, *query*) weights
- Each head will output a vector Z_i



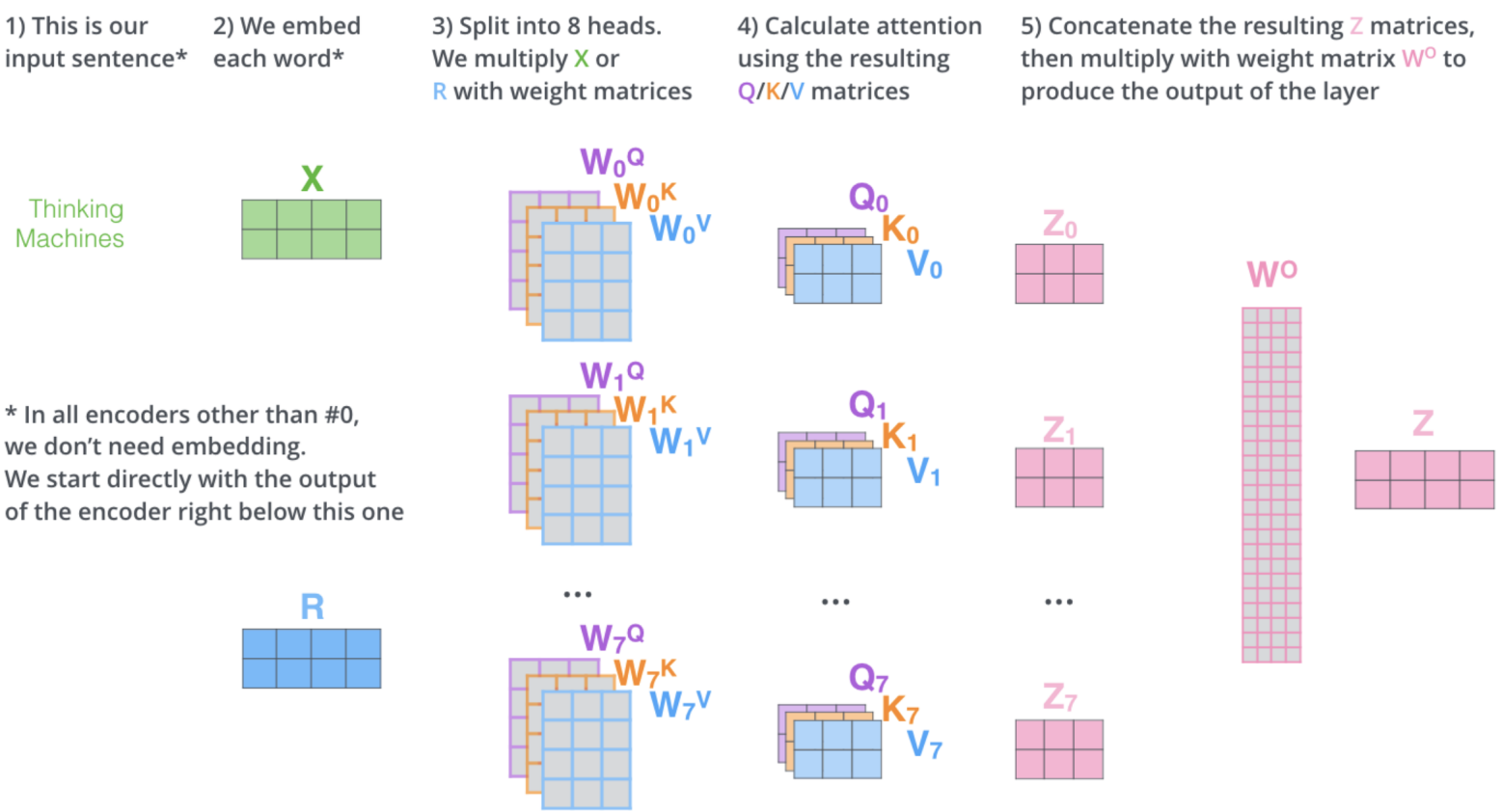


Multiply with weight matrix to reshape

- Gather all the outputs $Z_1, \ldots, Z_k$
- Multiply with a weight matrix to reshape
- Then pass to the next fully connected layer

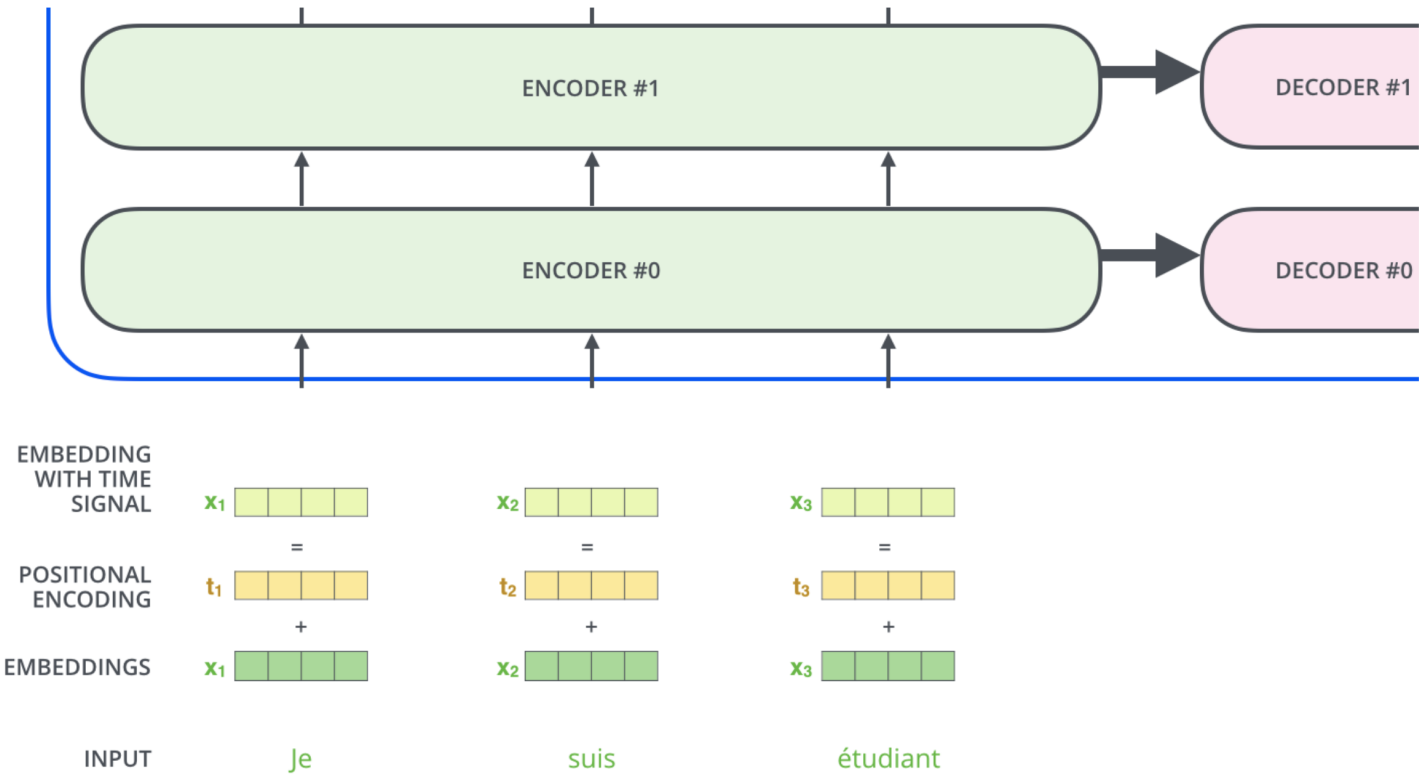


Overall architecture



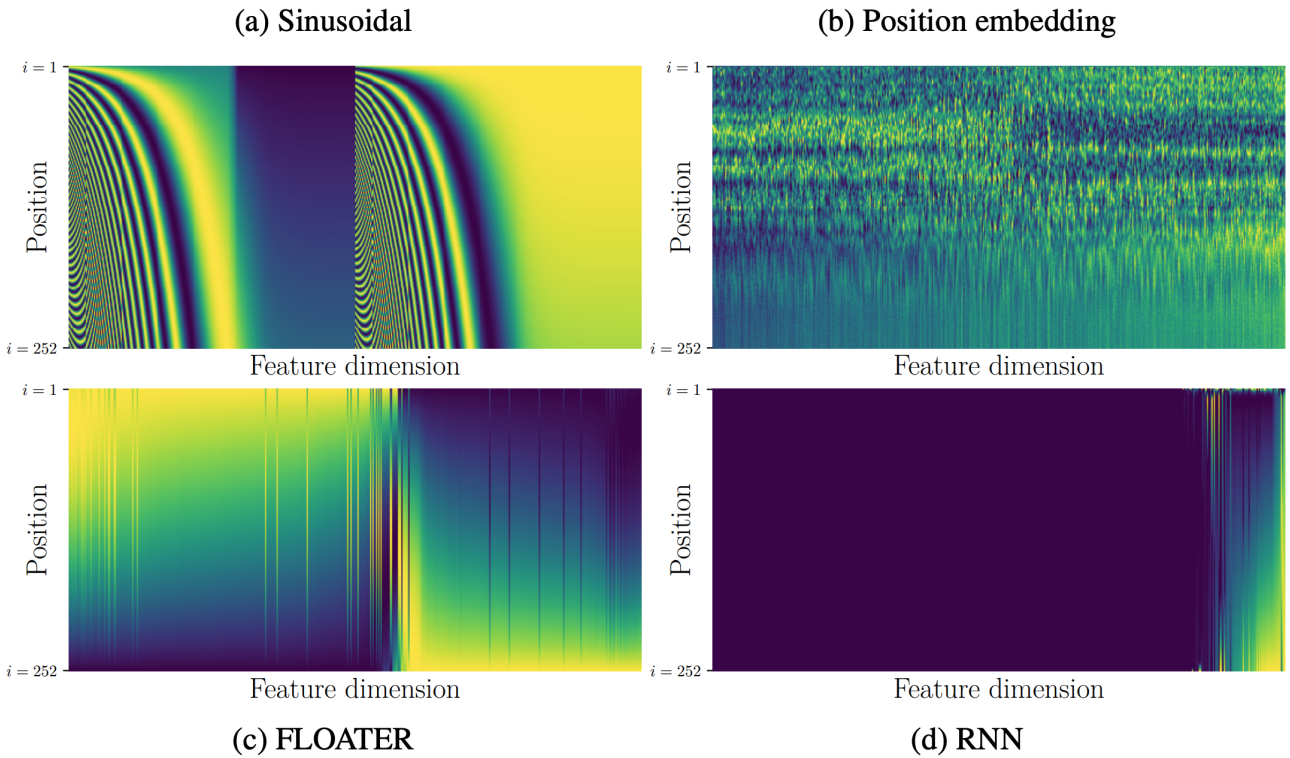
Position encoding

- The above architecture ignores the sequential information
- Add a position encoding vector to each $\boldsymbol{x}_i$ (according to i)



Types of position encoding

- Sin/cosine functions with different wavelengths (used in the original Transformer)
  - smooth, parameter-free, inductive
- Position embedding: learn a latent vector for each position
  - non-smooth, data-driven (learnable), non-inductive
- Neural ODE embedding (Liu et al., 2020)
  - smooth, data-driven (learnable), inductive



Conclusions

- A brief introduction of RNN and Transformer.

Questions?